制約充足問題における体系的探索手法の効率化についての検討

永 井 保 夫*

制約充足問題は人工知能や画像解析の分野をはじめ、グラフの問題やパズルなどの探索問題、いわゆる組み合わせ問題を対象として研究がおこなわれている。制約充足問題とは、変数の有限集合および各変数に対応する離散値の有限集合のドメインから値を選択し、すべての制約を満足するように各変数に対して値を割り当てる問題である。制約とは適用対象の構成要素およびその属性間で成立する関係を宣言的に記述したものである。

制約充足問題における従来の解法としては、バックトラックを基本とした探索による方法や整合化手法などに代表される前処理を用いる方法が代表的である。この場合には、解法としては探索空間の全域を対象としてすべての可能性を探索するもので、アルゴリズムの完全性を保証している。完全性とは、探索空間に解が存在すれば、必ず見つけられることと、解がなければ必ず存在しないことを保証するものである。

このような体系的探索法を用いる解法では、生成検査法やバックトラック法などに基づき解が求められる。特に、すべての解を求める場合には、探索木全体の探査が要求される。バックトラックに基づいた探索では、無駄なバックトラックが頻発してしまい効率が悪い。特に、全解を求めたい場合には、探索木全体をたどることになってしまうので、無駄な分岐を避けることが望ましい。

本論文では、まず、バックトラックによる効率改善手法として、知的バックトラックの一種であるバックジャンピングやバックトラックと整合化手法を組み合わせたフォワードチェッキングといった無駄な探索枝を刈り込む手法について説明する。次に、制約充足問題の代表例である地図の色塗り問題へ適用し、その結果について示す。

キーワード：制約充足問題，探索，バックトラック，アルゴリズム，効率化

Towards an Improvement of Systematic Search Methods for Constraint Satisfaction Problems

Yasuo NAGAI

In this paper, we describe and discuss improvement methods of systematic search for constraint satisfaction problems. Effectiveness of these improvement methods is shown using application of these methods to map coloring problem.

Keyword：constraint satisfaction problem, search, backtrack, algorithm, efficiency

*東京情報大学総合情報学部情報システム学科
Tokyo University of Information Sciences, Faculty of Informatics, Department of Information Systems

2004年11月27日受理

1　はじめに

制約充足問題は人工知能や画像解析の分野をはじめ、グラフの問題やパズルなどの探索問題、いわゆる組み合わせ問題を対象として研究がおこなわれている[1][2][3][4][7][8][9]。制約充足問題とは、変数の有限集合および各変数に対応する離散値の有限集合のドメインから値を選択し、すべての制約を満足するように各変数に対して値を割り当てる問題である。制約とは適用対象の構成要素およびその属性間で成立する関係を宣言的に記述したものである。制約充足問題における従来の解法としては、バックトラックを基本とした探索による方法や整合化手法などに代表される前処理を用いる方法が代表的である［1][4][6]。この場合には、解法としては探索空間の全域を対象としてすべての可能性を探索するもので、アルゴリズムの完全性を保証している［5]。完全性とは、探索空間に解が存在すれば、必ず見つけられることと、解がなければ必ず存在しないことを保証するものである。このような体系的探索法を用いる解法では、生成検査法やバックトラック法などに基づき解が求められる。特に、すべての解を求める場合には、探索木全体の探査が要求される。バックトラックに基づいた探索では、無駄なバックトラックが頻発してしまい効率が悪い。特に、全解を求めたい場合には、探索木全体をたどることになってしまうので、無駄な分岐を避けることが望ましい［4][5][6]。

本論文では、まず、バックトラックによる効率改善手法として、知的バックトラックの一種であるバックジャンピング［1][4][5][6］やバックトラックと整合化手法を組み合わせたフォワードチェッキングといった無駄な探索枝を刈り込む手法について説明する。次に、制約充足問題の代表例である地図の色塗り問題へ適用し、その結果について示す。

2　制約充足問題

2.1　制約充足問題の定義

制約充足問題とは、次のような変数の有限集合および各変数に対応する離散値の有限集合のドメインから値を選択し、すべての制約を満足するように各変数に対して値を割り当てる問題である。制約とは適用対象の構成要素およびその属性間で成立する関係を宣言的に記述したものである。

- 変数集合：$V=\{v_1,\ldots,v_n\}$，各 v_i は互いに独立な変数。
- ドメイン: 離散値の有限集合$D_i=\{d_{i_1},\ldots,d_{i_m}\}$，$i=1,\ldots,n$，$d_{ij}$ $(j=1,\ldots,m)$ は数値または記号値をあらわす。各変数 v_i には D_i の要素である値 d_{i_m} が割り当てられる。
- 制約集合：$C=\{c_1,\ldots,c_l\}$．ここで、各 c_i は制約で、それは次のような等式
  $(v_1, v_2,\ldots,v_n)=\langle$直積$D_1\times D_2\times\ldots\times D_n$ の部分集合$\rangle$
  の表現形式をとる。

制約充足問題における従来の解法としては、バックトラックを基本とした探索による方法や整合化手法などに代表される前処理を用いる方法が代表的である。ところで、n 個の変数に対する制約、つまり n 項制約は 2 項制約により表現できるので、今後は、 2 項制約のみを対象とした制約充足問題を考える。

制約充足問題では、変数を表すノードと制約によりラベル付けされたアーク（エッジ）からなるグラフとして表現される。このようなグラフは制約ネットワークとして静的な問題の記述に適した知識表現とみなれる。ネットワークを用いたアプローチの利点は、知識の構造化が容易であり、知識の無矛盾性を効率的に管理できることである。制約ネットワーク$\mathcal{R}$は、n 個の変数 $v_1, v_2,\ldots,v_n$ を要素とする集合 V、各変数に対するドメイン$D_1, D_2,\ldots,D_n$ を要素とする集合 D および $c_1, c_2,\ldots,c_n$ を要素とする制約（n 項関係）集合から構成され、三つ組（V, D, C）により表現される。n 個の変数 $v_1, v_2,\ldots,v_n$ に対する制約 $c(v_1, v_2,\ldots,v_n)$ とは、n 個の集合 $\{D_1, D_2,\ldots,D_n\}$ に対して成り立つ関係（つまり n 項関係）を表し、無矛盾な変数値の直積 $D_1\times D_2\times\ldots\times D_n$ の部分集合である。例えば、 2 項制約ネットワークはすべての制約が2 項関係である（高々 2 個の変数からなる）ネットワークである。それから、変数に対してドメインから値が割り当てられるとき、変数が具体化されたという。変数集合の具体化とは、各変数のドメインからの値の割り当てを意味する。つまり、変数集合 $\{v_{i_1},\ldots,v_{i_k}\}$ の具体化とは、順序付けされたペアのタプル（$\langle v_{i_1}, a_{i_1}\rangle,\ldots,\langle v_{i_k}, a_{i_k}\rangle$）を表す。但し、各ペア（$\langle v, a\rangle$）は変数 v への値 a の割り当てを、a は v のドメインを示す。このような変数集合の具体化は（$v_1=a_1,\ldots,v_i=a_i$）と表現する。た

とえば、（⟨v_1, a_1⟩, . . . ,⟨v_i, a_i⟩）は $\bar{a}=(a_1, \ldots, a_i)$ と表す。

制約ネットワーク$\mathcal{R}=(V, D, C)$の解とは、すべての制約を満足するようなすべての変数の具体化を意味する。一般に、制約充足（制約を満足させる）とは、このような制約ネットワークを解くことであり、すべての制約が満足されるようにネットワーク中のすべての変数に対して値を求めることに相当し、単解または全解を求めることを意味する。

2.2 体系的探索手法を用いる解法

体系的探索法を用いる解法では、生成検査法やバックトラック法などに基づき解が求められる。特に、すべての解を求める場合には、探索木全体の探査が要求される。バックトラックに基づいた探索では、無駄なバックトラックが頻発してしまい効率が悪い。特に、全解を求めたい場合には、探索木全体をたどることになってしまうので、無駄な分岐を避けることが望ましい。そこで、バックトラックの効率改善手法として、知的バックトラックの一種であるバックジャンピングやバックトラックと整合化手法を組み合わせたフォワードチェッキングといった無駄な探索枝を刈り込む手法が提案されている。

以下では、状態空間と代表的な体系的探索手法（バックトラック、バックジャンピング、フォワードチェッキング）についてそれぞれ説明する。

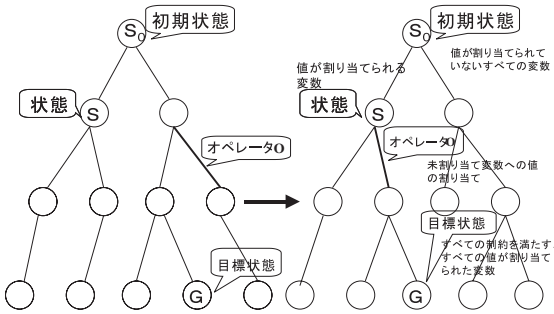
2.2.1 状態空間

制約充足問題の状態空間は、図１のように表現される。

従来の探索に基づく問題解決における状態空間表現と制約充足問題における状態空間表現の対応関係を図２に示す。

・状態集合 S
　値が割り当てられる変数
・オペレータ集合 O
　未割り当ての変数への値の割り当て
・初期状態 $s_0 \in S$
　値が割り当てられていないすべての変数
・ゴール状態 $S_g \subseteq S$
　すべての制約を満たす、すべての値が割り当てられた変数

図１　制約充足問題の状態空間表現



(a) 探索による問題解決での状態空間の表現 (b) 制約充足問題での状態空間の表現

図２　探索による問題解決および制約充足問題で取り扱う状態空間

状態空間は、図２の（a）のように、初期状態を根（ルート）、各状態を節点（ノード）、オペレータを枝（アーク）、そして最終状態を複数の葉（リーフ）としてもつ木（ツリー）として表現される。制約充足問題は、このような状態空間を表す木（この木を探索木とよぶ）における探索問題とみなすことができる。探索木の各枝が変数への値の割り当てを表し、その深さ（レベル）が変数の数を、分岐数が変数に対応するドメインのサイズをそれぞれ表す。制約充足問題の探索木（探索空間）は、値の割り当てられる変数の順序（variable ordering) によって決定される。

2.2.2 生成検査法（generate & test）

探索を用いた解法で最も単純なアプローチは、生成検査法である。生成検査法では、各変数に対する値の割り当ての可能なすべての組み合わせ（直積$D_1 \times D_2 \times \ldots \times D_n$）を生成し、制約が満足されるかどうかがチェックされる。

2.2.3 バックトラック（backtrack）

バックトラックは、以下の手順でおこなわれる。

Step 1：与えられた順序で変数に対して順番に値を割り当てることによって、変数のドメインD の要素を機械的に調べていく。

Step 2：割り当ての対象となっている変数を含んだ制約条件は、そこで使われているすべての変数に対する値の割り当てが行われると、すぐにチェックされる。

Step 3：制約条件が満たされなければ、ドメインに値が残っていてまだ割り当てられていない変数がある

場合には、その値を変数に割り当てる。

Step 4 ：割り当てる値がなくなれば後戻りし、別の変数への値の割り当てをおこなう。

バックトラックでは、深さ優先探索を用いた木の探索がおこなわれる。バックトラックは2つのフェーズから構成される。1番目は前向きフェーズで、変数が順番に選択され、部分解は次の変数に無矛盾な値を割り当てることで求められる。2番目は後ろ向きのフェーズであり、現時点の変数に対して制約に違反しない解が存在しない場合には、前に値が割り当てられた変数に戻る。なお、変数の選択順序については、静的順序と動的順序の2つがある。静的順序は、探索が始まる前に、すでに決定されている変数の順序であり、通常のバックトラックで使わることが多い。動的順序は、探索時に次の変数の選択順序を現在の探索状況に基づき動的に決定するものである。

バックトラックでは、制約チェックによって解が生成できなくなった時点で、ひとつの変数への値の割り当てをリジェクトして、探索空間の一部分（探索しても無駄な部分）を取り除いている。しかしながら、変数の数が増加すると、解を求めるために必要となる操作が多くなるため、処理が非効率になってしまう。

そこで、バックトラックの効率改善のために、変数の順序付けと変数への値の割り当て順序についてのヒューリスティックス（heuristics）が利用されている。前者の変数の順序付けについては制約違反をいかに早く発見するかという点から、後者の変数への値の割り当て順序については制約違反を避けるという点から提案されている。

まず、変数の順序付けについてのヒューリスティックスであるが、以下の2つが使われている。

・最も多くの制約が与えられた変数を最初に選択する（first-fail principle として知られており、最も利用される）。

　つまり、これはドメインの値の要素数が最も少ない変数を選択する。これにより、探索木のサイズを小さくできる傾向がある。

・最も多くの制約に関連している変数を選択する。

次に、変数への値の割り当て順序についてのヒューリスティックスであるが、以下の2つが用いられている。

・最も制約への関係が少ない（違反の原因とならない）値を選択する。

・最も期待値が高い値を選択する

　ただし、この期待値は、問題固有の知識を利用することが必要である。

それから、バックトラックアルゴリズムの詳細は、図3に示される。

```
procedure BACKTRACK
入力：制約ネットワーク ℛ = (V, D, C)
出力：解又はネットワークが不整合（解が存在しない）との
      通知
1   begin
2     i ← 1
3     D'_i ← D_i
4     while 1 ≤ i ≤ n
5       SELECT-VALUEから求めた値を v_i とする
6       if v_i is null
7         i ← i − 1
8       else
9         i ← i + 1
10        D'_i ← D_i
11    if i = 0
12      return “inconsistent”
13    else
14      return 解 {v_1, . . . , v_n}
15  end
```

図3　BACKTRACK 手続き

2.2.4　バックトラックの効率改善手法

（1）バックジャンピング（backjump）

バックジャンピングは、探索時に同じ行き止まりを繰り返し再発見してしまうという、バックトラックの問題点を改善するために考えられた手法である。バックジャンピングでは、すでに値の割り当てられた変数間の制約をチェックし（これをLook Back という）、制約が満たされない場合には、その原因を見つけるために状況を分析し、最も最近に制約に違反している変数への値の割り当てを変更する。

バックジャンピングでは、探索時に、制約に違反している原因に直接移動することで、訪問する探索木のノード数を最小化し、整合化チェックの数を減少させ、探索効率の改善を図る。

なお、バックジャンピングアルゴリズムの詳細は、図5に示される。

（2）フォワードチェッキング（forward checking）

フォワードチェッキングは、一時的に変数に値を割

```
procedure SELECT-VALUE
入力：制約ネットワーク ℛ = (V, D, C)
出力：ā_{i-1} と整合な D'_i の要素である値を返す
1  begin
2    while D'_i が空ではない
3      a ∈ D'_i から任意の要素を選択し、D'_i から a を削除する
4      if 変数 v_1 から v_{i-1} までの値の割り当て（ā_{i-1}）が
         変数 v_1 への値 a の割り当て（v_i = a）と整合関係にあれば
5        return a
6    return null
7  end；
```

図 4　SELECT-VALUE 手続き

```
procedure BACKJUMPING
入力：制約ネットワーク ℛ = (V, D, C)
出力：解又はネットワークが不整合（解が存在しない）の通知
1  begin
2    i ← 1
3    D'_i ← D_i
4    latest_i ← 0
5    while 1 ≤ i ≤ n
6      SELECT-VALUE-GBJで求められた値を v_i とする
7      if v_i is null
8        i ← latest_i
9      else
10       i ← i + 1
11       D'_i ← D_i
12       latest_i ← 0
13   if i = 0
14     return “inconsistent”
15   else
16     return 解 {v_1, . . . , v_n}
17 end
```

図 5　Backjumping アルゴリズム

り当てておき、この割り当てと制約違反により競合すると考えられる将来の変数への値の割り当てを除去する。これにより、値が割り当てられていない変数のドメインに制限を与え、将来起こる可能性がある制約違反を避けることができる。つまり、フォワードチェッキングは過去および現在の変数を基準として、未来の変数に割り当てる候補となる値のうちで、無効なものを削除する方法である。もし、候補となる値がすべて削除されてしまうような変数がみつかれば、現在の枝の探索は打ち切られる。

なお、フォワードチェッキングの詳細なアルゴリズム

```
procedure SELECT-VALUE-GBJ
入力：制約ネットワーク ℛ = (V, D, C)
出力：整合な値または null
1  begin
2    while D'_i が空ではない
3      a ∈ D'_i から任意の要素を選択し、D'_i から a を削除する
4      consistent ← true
5      k ← 1
6      while k < i and consistent
7        if k > latest_i
8          latest_i ← k
9        if 変数 v_1 から v_k までの値の割り当て（ā_k）が
           変数 v_i への値 a の割り当て（v_i = a）と整合関係になれば
10         consistent ← false
11       else
12         k ← k + 1
13       if consistent
14         return a
15   return null
16 end；
```

図 6　SELECT-VALUE-GBJ 手続き

```
procedure FORWARD-CHECKING
入力：制約ネットワーク ℛ = (V, D, C)
出力：解または null
1  begin
2    D'_i ← D_i for 1 ≤ i ≤ n
3    i ← 1
4    while 1 ≤ i ≤ n
5      SELECT-VALUE-FORWARD-CHECKINGで求められた値を v_i とする
6      if v_i が null 値である
7        i ← i − 1
8        v_i に最後に値が割り当てられる前に、各 D'_k (k > i) をリセットする
9      else
10       i ← i + 1
11   if i = 0
12     return “inconsistent”
13   else
14     return 解 {v_1, . . . , v_n}
15 end；
```

図 7　フォワードチェッキングアルゴリズム

ムは、図 7 に示される。

3　制約充足問題への体系的探索法の効率化手法の適用

3.1　地図の色塗り問題

地図の色塗り問題とは、地図といくつかの色が与え

```
procedure SELECT-VALUE-FORWARD-CHECKING
入力：制約ネットワーク ℛ = (V, D, C)
出力：整合な値または null
1   begin
2     while D'_i が空ではない
3       D'_i から任意の要素 a を選択し、D'_i から a を削除する
4       for i < k ≤ n となるようなすべての k
5         for D'_k の要素であるすべての値 b
6           if 変数 v1 から vk までの値の割り当て（ā_k）が変
              数 vi への値 a の割り当て（vi = a）ならびに変
              数 vk への値 b の割り当て（vk = b）と整合関
              係になれば
7           D'_k から b を削除する
8           if D'_k が空
9           a が選択される前に、各 D'_k (i < k ≤ n) をリ
              セットする
10          else
11          return a
12    return null
13  end；
```

図 8　SELECT-VALUE-FORWARD-CHECKING 手続き

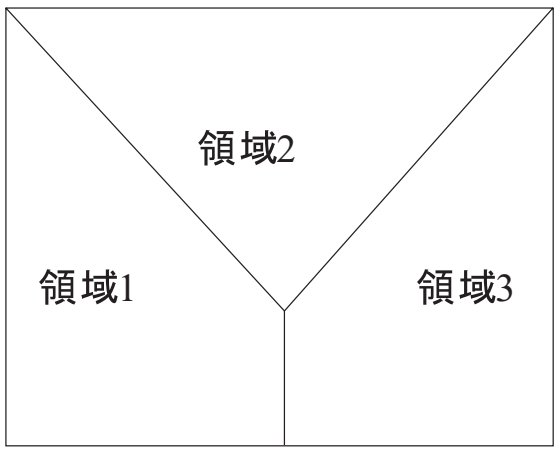


図 9　地図の色塗り問題

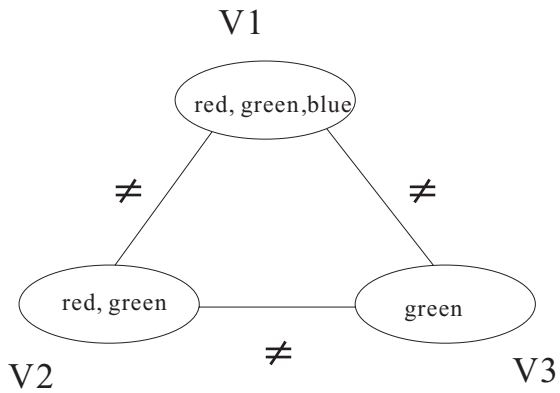


図10　地図の色塗り問題の制約ネットワーク表現

られた時、隣接した領域では同じ色にならないように、それぞれの領域に色を割り当てていく問題である（図9）。

制約充足問題としての定式化は以下のとおりである。

・変数集合：$V = \{v_1, \ldots, v_n\}$，各 v_i は互いに独立な変数。

$V = \{v_1$（＝領域１），v_2（＝領域２），v_3（＝領域３）$\}$

・ドメイン：離散値の有限集合$D_i = \{d_{i_1}, \ldots, d_{i_m}\}$，$i = 1, \ldots, n$，$d_{ij}$ $(j = 1, \ldots, m)$ は数値または記号値をあらわす。

$D_1 = \{red, green, blue\}$，$D_2 = \{red, green\}$，$D_3 = \{green\}$

・制約集合：$C = \{c_1, \ldots, c_l\}$．ここで、各 c_i は、v_i と v_j と隣接していれば、v_i と v_j は同じ色を塗れない $(v_i \neq v_j)$ という制約をあらわす。

$c_1 = R_{12} = \{(red, green), (green, red), (blue, red), (blue, green)\}$

$c_2 = R_{23} = \{(red, green)\}$

$c_1 = R_{13} = \{(red, green), (blue, green)\}$

図 9 の地図の色塗り問題の制約ネットワークを図10に示す。

3.2　体系的探索手法を用いる解法

体系的探索法を用いる解法では、生成検査法やバックトラック法などに基づき解が求められる。特に、すべての解を求める場合には、探索木全体の探査が要求される。

バックトラックに基づいた探索では、無駄なバックトラックが頻発してしまい効率が悪い。特に、全解を求めたい場合には、探索木全体をたどることになってしまうので、無駄な分岐を避けることが望ましい。そこで、バックトラックの効率改善手法として、知的バックトラックの一種であるバックジャンピングやバックトラックと整合化手法を組み合わせたフォワードチェッキングといった無駄な探索枝を刈り込む手法が提案されている。

以下では、状態空間と代表的な体系的探索手法（バックトラック、バックジャンピング、フォワードチェッキング）についてそれぞれ説明する。

3.2.1 状態空間

図10で示される地図の色塗り問題が表現される制約ネットワークを考えてみよう。この制約ネットワークに対して、変数の順序（$v_1 \rightarrow v_2 \rightarrow v_3$）が与えられた場合の探索木が図11で示される。図12は、変数の順序（$v_2 \rightarrow v_3 \rightarrow v_1$）が与えられた場合の探索木を、図13は、変数の順序（$v_3 \rightarrow v_2 \rightarrow v_1$）が与えられた場合の探索木をそれぞれ示している。

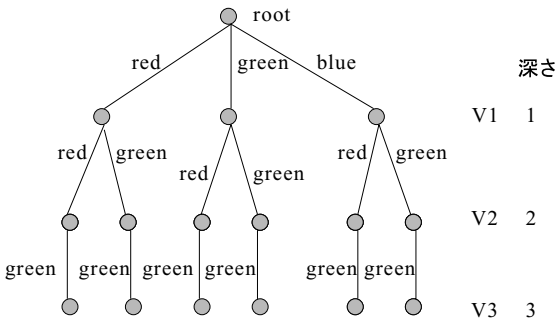


図11 地図の色塗り問題の探索空間（1）

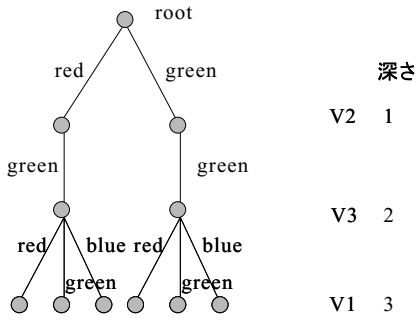


図12 地図の色塗り問題の探索空間（2）

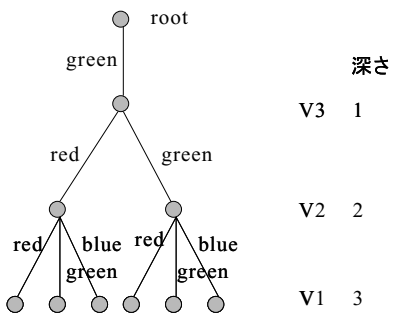


図13 地図の色塗り問題の探索空間（3）

3.2.2 生成検査法

図14は、図10で示される地図の色塗り問題を生成検査法を用いた場合に作成される探索木を示している。変数 v_1, v_2, v_3 に対する割り当て可能なすべての値の組み合わせは、$\{(\langle v_1, red\rangle, \langle v_2, red\rangle, \langle v_3, green\rangle), (\langle v_1, red\rangle, \langle v_2, green\rangle, \langle v_3, green\rangle), (\langle v_1, green\rangle, \langle v_2, red\rangle, \langle v_3, green\rangle), (\langle v_1, green\rangle, \langle v_2, green\rangle, \langle v_3, green\rangle), (\langle v_1, blue\rangle, \langle v_2, red\rangle, \langle v_3, green\rangle), (\langle v_1, blue\rangle, \langle v_2, green\rangle, \langle v_3, green\rangle)\}$ となる。制約のチェックは、これらの組み合わせがすべて生成された後におこなわれる。生成検査法はしらみ潰し法の一種であり、取り扱う問題のサイズに依存するため、非効率である。

3.2.3 バックトラック

以下では、地図の色塗り問題のバックトラックによる木探索処理が変数の順序により異なる、つまり探索効率に影響を及ぼすことを見てみる。

図15は、静的な変数の順序（$v_1 \rightarrow v_2 \rightarrow v_3$）が与えられた場合のバックトラックにより作られる探索木を示している。この場合、1つの解を見つけるのに、探索木では節点12個生成され、制約のチェックが10回行われている。図16は、静的な変数の順序（$v_2 \rightarrow v_3 \rightarrow v_1$）が与えられた場合のバックトラックにより作られる探索木を示している。ここでは、6個の節点が生成され、6回の制約チェックが行われている。図17は、静的な変数の順序（$v_3 \rightarrow v_2 \rightarrow v_1$）が与えられた場合のバックトラックにより作られる探索木を示している。探索木では、6個の節点が作られ、5回の制約チ

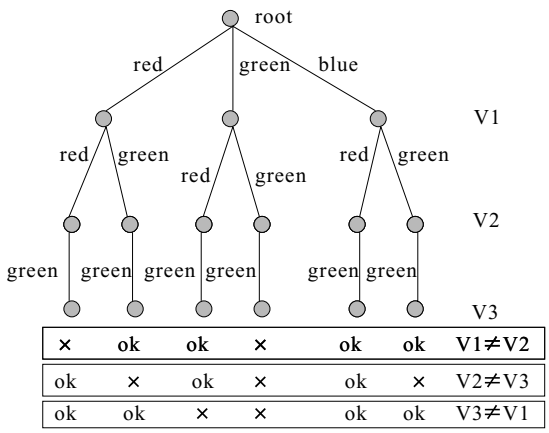


図14 生成検査法

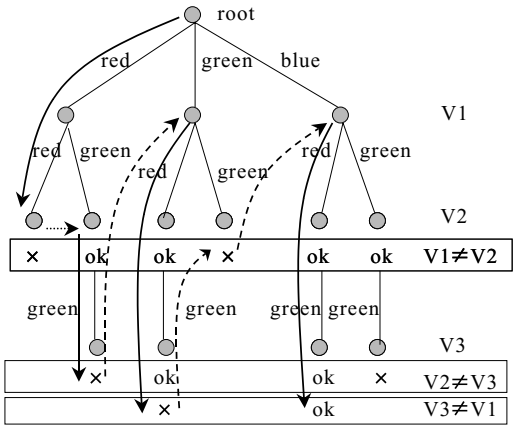


図15　バックトラック（1）

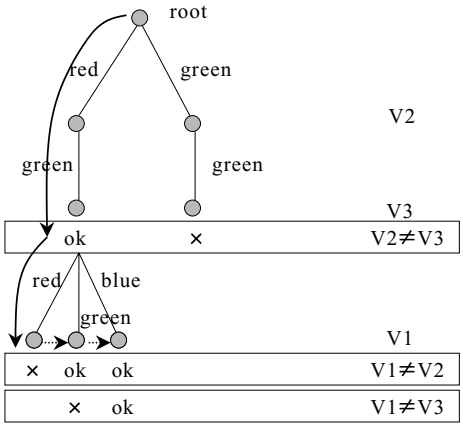


図16　バックトラック（2）

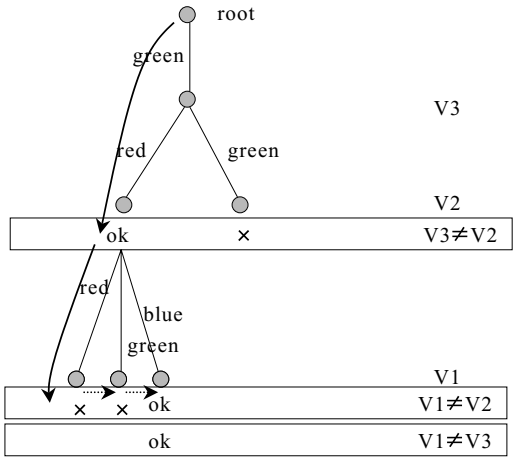


図17　バックトラック（3）

ェックが行われている。以上のように、与えられる変数の順序によってバックトラックの処理効率が改善することがわかる。

3.2.4　バックトラックの効率改善手法

（1）バックジャンピング

図10に対してバックジャンピングを適用する。主要な処理の流れは、バックトラックに基づいており、図18のとおりである。

1．部分解（$\langle v_1, red\rangle, \langle v_2, red\rangle$）を求める（（1）の部分に対応）。

2．1の部分解への制約チェックの結果、制約が満たされないのでバックトラックして、変数 v_2 へ *green* を割り当てる。その結果、部分解（$\langle v_1, red\rangle$, $\langle v_2, green\rangle$）を求める（（2）に対応）。

3．2の部分解に対して、さらに変数 v_3 へ *green* を割り当て、その結果、（$\langle v_1, red\rangle, \langle v_2, green\rangle, \langle v_3, green\rangle$）を得る（（3）に対応）。

4．3に対する制約チェックの結果、制約が満たされないので、バックトラックして新たに部分解（$\langle v_1, green\rangle$）を求める（（4）に対応）。

5．4に対して、v_2 へ *red* の割り当て、さらに、v_3 へ *green* の割り当てをおこなう（（5）に対応）。

6．5で求められた割り当てに対する制約チェックの結果、制約が満たされないので、変数 v_3 と制約関係がある変数を変数 v_3 の競合集合に記録する。ここでは、v_3 の競合集合は $\{v_1\}$ となる（（6）に対応）。

7．変数 v_3 に対して、既に値が割り当てられている変数と整合性がとれる値が見つからないので、v_3 の競合集合の中で最も最近に値が割りてられた変数 v_1 を選択する（（6）に対応）。

8．変数 v_1 に移動し、通常のバックトラックをおこなう。ここでは、点線矢印で示されているように、変数 v_1 へ移動する（（6）に対応）。

9．新たな変数 v_1 への値の割り当てにより求められた部分解（$\langle v_1, blue\rangle$）に対して、変数 v_2 へ値を割り当て、制約チェックをおこなう（（7）に対応）。

10．制約（$v_1 \neq v_2$）が満足されているので、部分解（$\langle v_1, blue\rangle, \langle v_2, red\rangle$）が生成され、これに対して変数 v_3 への *green* の割り当てをおこなう（（7）

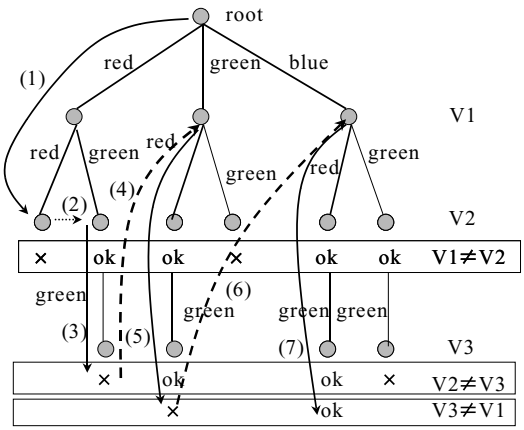


図18　バックジャンピング

に対応）。

11. 以上で求められた変数 v_1, v_2, v_3 への割り当てに対して制約チェックをおこない、その結果、解（$\langle v_1, blue\rangle, \langle v_2, red\rangle, \langle v_3, green\rangle$）が得られる。

（2）フォワードチェッキング

図10の制約ネットワークで表現される地図の色塗り問題にフォワードチェッキングアルゴリズムを適用すると以下のようになる。

1. **変数 v_1 に対して、値 r（$=red$）を割り当てる。**

図19の左側では、制約ネットワークが示されている。ここでは、r への下線はノード v_1 のドメイン $\{r(=red), g(=green), b(=blue)\}$ から r が選択されたことを表している。図19の右側では、フォワードチェックにより作られた探索木が示されている。

2. **上記の値が割り当てられた変数を用いて整合化手法（アーク整合）を実行する（これを制約の伝播をおこなうという）。**

図20の制約ネットワークのノード v_1 に隣接しているノード v_2 と v_3 に対してアーク整合をチェックする。ここでは、ノード v_1 に割り当てられた値 r と矛盾するノード v_2 のドメインの値ならびに v_3 のドメインの値を取り除く。その結果、ノード v_2 ではそのドメインから r が取り除かれ、g が残る。

3. **変数 v_2 に対して、値 g（$=green$）を割り当てる。**

図21の左側に示される制約ネットワークでは、ノード v_2 に対して値g が選択されたことが g への下線として表現されている。図21の右側は対応して作られた探索木を示している。

4. **値 g が割り当てられた変数 v_2 を用いて整合化手法（アーク整合）を実行する。**

図22の制約ネットワークのノード v_2 に隣接しているノード v_1 と v_3 に対してアーク整合をチェックする。

ここでは、ノード v_2 に割り当てられた値 g と矛盾するノード v_1 のドメインの値および v_3 のドメインの値を取り除く。その結果、ノード v_3 ではそのドメインから g が取り除かれ、要素が空となるので、ノード整合が成立しないことがわかる。

5. **上記の結果より後戻りして、変数への値の割り当てをやり直す。**

ノード v_2 への値の割り当てを再度行なおうとするが、候補となる値がないので、ノード v_1 への値の割り当てをおこなう。ここでは、ノード v_1 に対して値 g を割り当てる。図23の左図に示される制約ネットワークにおけるノード v_1 のドメインの値 g への下線は、この割り当てを表している。

6. **値 g が割り当てられた変数 v_1 を用いて整合化手法（アーク整合）を実行する。**

図24に示される制約ネットワークのノード v_1 に隣接しているノード v_2 および v_3 に対してアーク整合をチェックする。このチェックでは、ノード v_2 に割り当てられた値 g に対して矛盾するノード v_1 のドメインの値ならびに v_3 のドメインの値を取り除く。その結果、ノード v_2 のドメインとノード v_3 のドメインから値 g がそれぞれ取り除かれる。最終的に、ノード v_3 のドメインの要素が空となるので、ノード整合が成り立たないことがわかる。

7. **上記の結果より後戻りして、変数への値の割り当てをやり直す。**

ノード v_1 への値の割り当てを再度行ない、値 b（$=blue$）を候補とする。図25の左図に示される制約ネットワークにおけるノード v_1 のドメインの値 b への下線は、この割り当てを表している。図25の右側が対応して作成された探索木を表している。

8. **値 b が割り当てられた変数 v_1 を用いて整合化**

手法（アーク整合）を実行する。

図26の制約ネットワークに示されるノード v_1 に隣接するノード v_2 ならびに v_3 に対してアーク整合をチェックする。ここでは、ノード v_1 に割り当てられた値 b に対して矛盾するノード v_1 のドメインの値および v_3 のドメインの値をそれぞれ取り除く。その結果は図27の左図に示され、対応して作成された探索木は図27の右図に示される。

9．**変数 v_2 に対して、値 r（$=red$）を割り当てる。**

図28に示される左側の制約ネットワークでは、ノード v_2 のドメインの値 r への下線はドメイン $\{r,g\}$ から値 r が選択されたことを示している。図28の右側には、対応して作られた探索木が示されている。

10. **値 r が割り当てられた変数 v_2 を用いて整合化手法（アーク整合）を実行する。**

図28に示される制約ネットワークのノード v_2 に隣接するノード v_1 と v_3 に対してアーク整合をチェックする。このチェックでは、ノード v_1 に割り当てられた値 r に対して矛盾する変数 v_1 のドメインの値および v_3 のドメインの値をそれぞれ取り除く。その結果、得られた制約ネットワークが図29の左図に、作成された探索木が図29の右図に示される。

以上のようにフォーワードチェックを適用した結果、求められた解は図29に示される。図29の左図が制

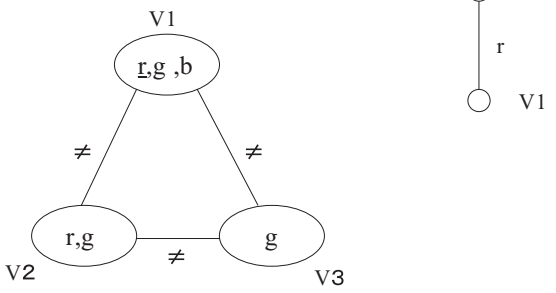


図19　フォワードチェッキング（1）

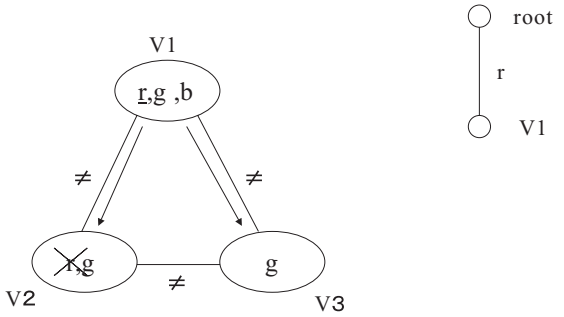


図20　フォワードチェッキング（2）

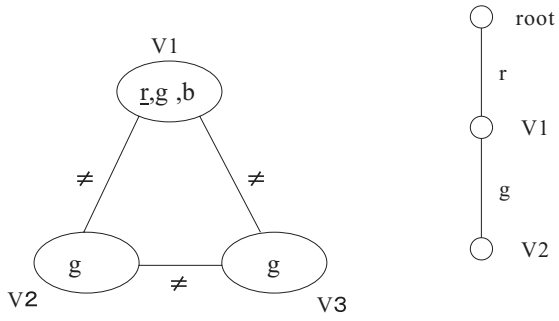


図21　フォワードチェッキング（3）

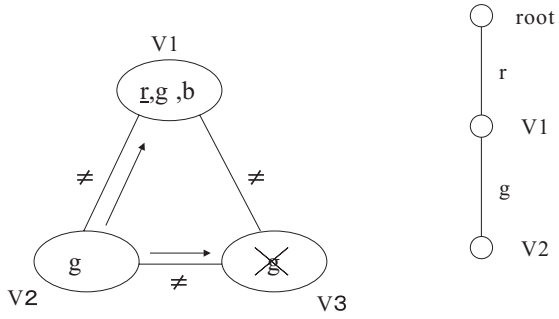


図22　フォワードチェッキング（4）

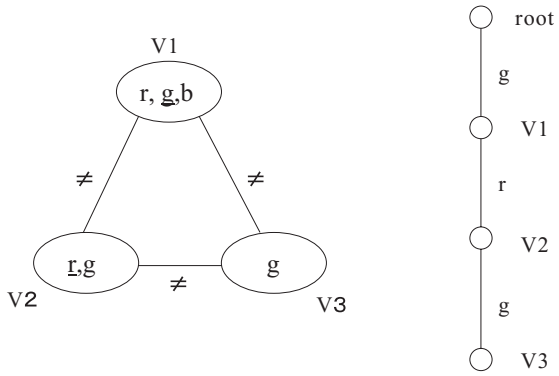


図23　フォワードチェッキング（5）

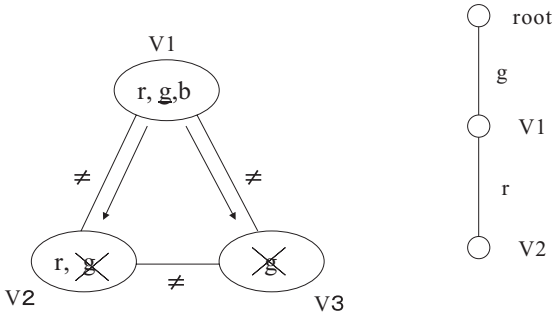


図24 フォワードチェッキング（6）

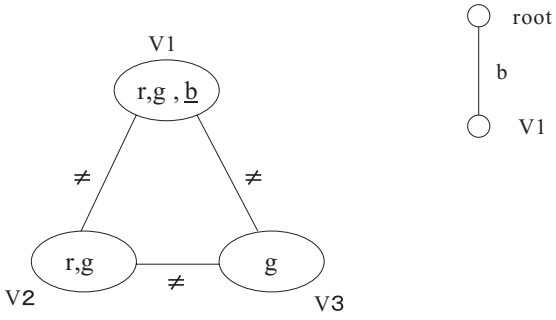


図25 フォワードチェッキング（7）

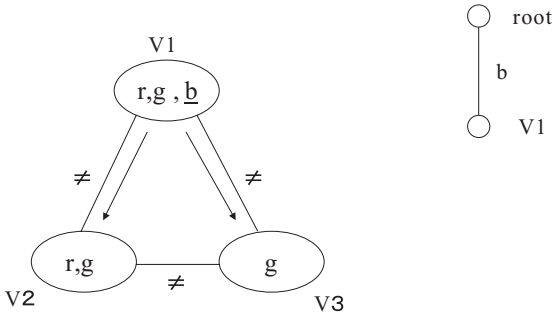


図26 フォワードチェッキング（8）

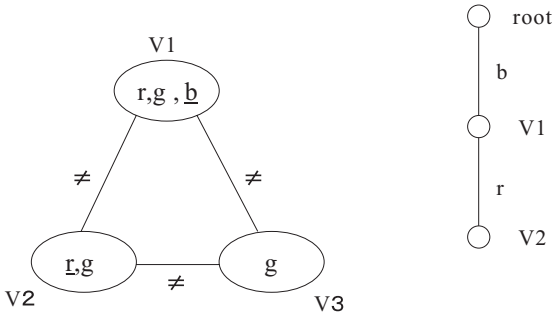


図27 フォワードチェッキング（9）

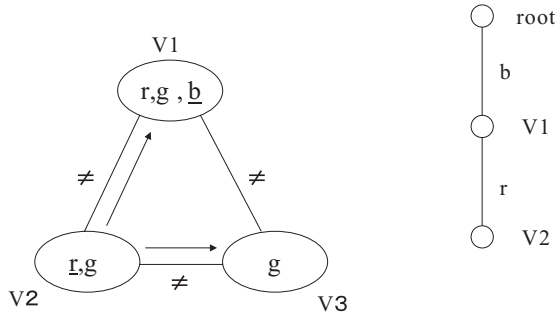


図28 フォワードチェッキング（10）

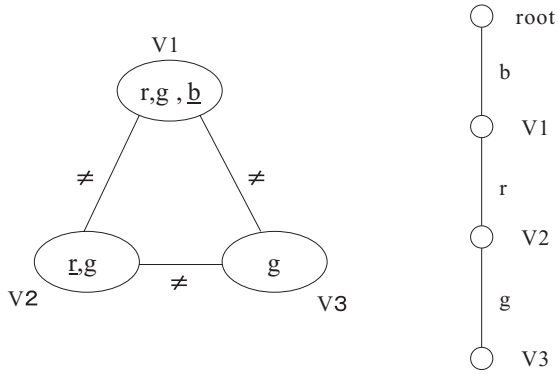


図29 フォワードチェッキング（11）

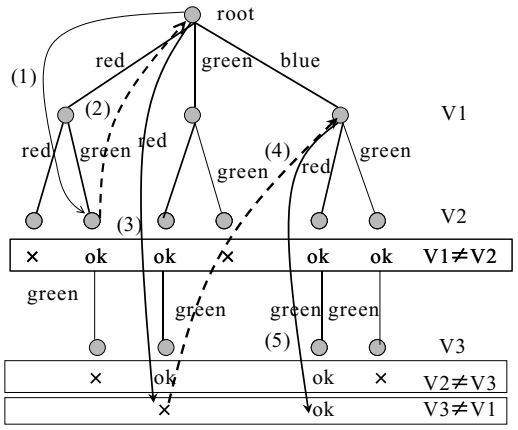


図30 フォワードチェッキングによる状態空間の探索

約ネットワークを、右図が探索木をそれぞれ表す。

最終的に、フォワードチェッキングによる状態空間の探索は図30のようになり、図15に示されるバックトラックと比較して効率化が実現されていることがわかる。

4　おわりに

本論文では、まず、バックトラックの効率改善手法として、知的バックトラックの一種であるバックジャンピングやバックトラックと整合化手法を組み合わせたフォワードチェッキングといった無駄な探索枝を刈り込む手法について述べた。次に、制約充足問題の代表例である地図の色塗り問題へ適用し、その結果について説明した。知的バックトラックは、体系的探索手法の効率化手法としては期待されている手法である。今後は、実用的な組み合わせ問題への適用を行い、有効性を評価していく予定である。

参考文献

［1］特集：制約充足問題の基礎と応用、人工知能学会誌、Vol.12, No.3（1997）.
［2］西原清一：整合ラベリング問題と応用、情報処理、Vol.31, No.4, pp.500-507（1990）.
［3］西原清一：制約充足問題（CSP）の基礎と動向、1991年度人工知能学会全国大会（第5回）チュートリアル資料B-1（1991）.
［4］Edward Tsang : *Foundations of Constraint Satisfaction*, Computation in Cognitive Science, Academic Press（1993）.
［5］Ian Miguel : *Dynamic Flexible Constraint Satisfaction and its Application to AI Planning*, Distinguished Dissertations, Springer（2003）.
［6］Rina Dechter : *Constraint Processing*, Morgan Kaufmann Publishers（2003）.
［7］Stuart Russell and Peter Norvig : *Artificial Intelligence A Modern Approach, Second Edition*, Prentice Hall Series in Artificial Intelligence, Pearson Education（2003）.
［8］David Pool, Alan Mackworth, and Randy Goebel : *Computational Intelligence: A Logical Approach*, Oxford Unversity Press（1998）.
［9］Alan Mackworth : Constraint Satisfaction, In *Encyclopedia of Artificial Intelligence（(ed.) Shapiro, S.C.), Vol.1*, pp.205-211, John Wiley & Sons, Inc.（1987）.